Inheritance

- Object Relationships
- supreme - superclass Object
- Extending a class
- Methods in subclass
- Using this reference
- Accessing superclass instance variables
- Using protected access
- Subclass construction
- Multiple constructors
- Method overloading

Read
Pages 302 - 311

Create another class for Savings Account

Must provide all the functionality of Account

In addition:

- **All saving account holders must maintain min-amount**
- **Must provide a method to compute interest**

Create from scratch

Extend Account class

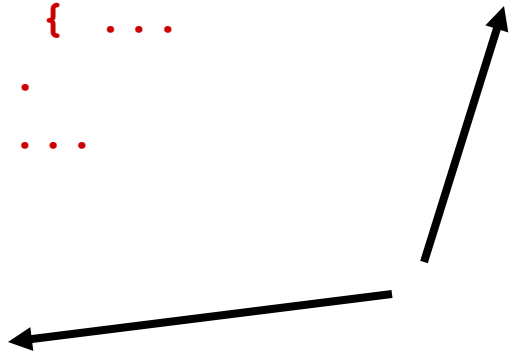
SAccount

Account

SAccount

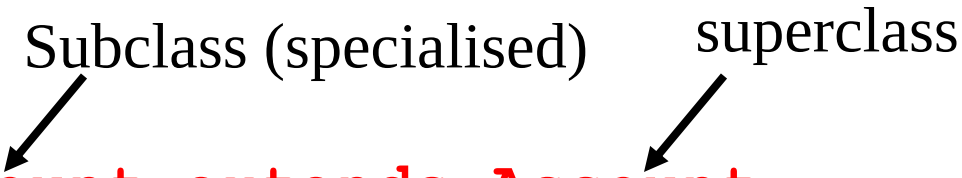
A new class SAccount

```
class SAccount  {
   public SAccount(String accountID, String accountName,
            double amount, double min) { ... // new
   public void addInterest(double rate) { ... // new
   public boolean withdraw(double amount) {... // changed
   public double getMinAmt() { ... // new

   public void deposit(double amount) { ...
   public boolean transfer(SAccount account, double amount) { ...
   public double getBalance() { ...
   public String getID() { ...
   public String getName() { ...

   private String name;
   private double balance;
   private String accID;
   private double minAmt; // new
}
```



Same as Account class

Could have used inheritance!




```
class SAccount extends Account
{
      new and redefined methods
      new instance variables;
}
```

- Add a new method addInterest(double rate)
- Redefine withdraw()
- add Instance variable minAmount
- write the constructor

Whenever I change superclass implementation it will be reflected in the subclasses. No worries !!!

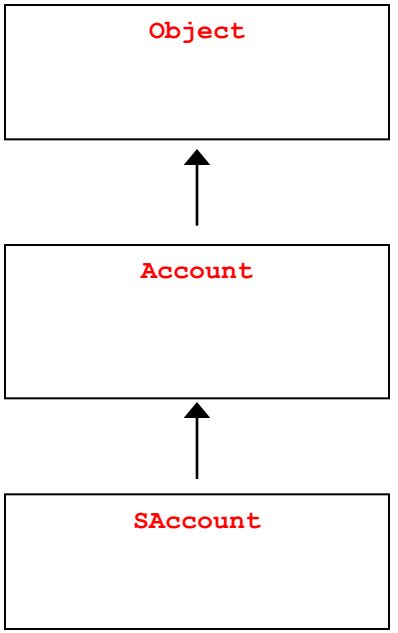
The supreme superclass Object

Every class (even one that does not use extends) is a subclass of Object.

Why ?

Object class has a small number of methods that are useful for all classes.

- equals()
- toString()
- clone()



Object Oriented Relationships

Two commonly used clauses in class relationship

- **is-a** ⟵ Inheritance relationship
- **has-a** ⟵ Composition relationship

A home is a house that has a family and a pet.

If House, Family and Pet are existing classes then in java we write

```
public class Home extends House
{
      Family inhabitants;
      Pet thePet;
}
```

Guess the relationship between the classes

a) `Manager` and `Employee`

b) `Project` and `Manager`

c) `Person` and `Student`

d) `Book` and `Author`

How would you write them in Java ?

A Re-look at Account class

public class Account {

public Account(…

public double getBalance(..

public String getID()..

public String getName()..

public void deposit(…

public void withdraw(…

public void print(…

public void transfer(…

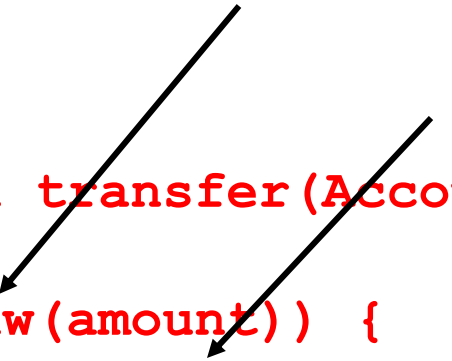
private String accID;

private double balance;

private String name;

Well Encapsulated

Note transfer uses other operations of the class - need not check again.



```
public boolean transfer(Account account,
                                    double amount) {
    if (withdraw(amount)) {
       account.deposit(balance);
       return true;
    }
    else return false;
}

void print()   {
  System.out.println("\nAccount ID = " + accID);
  System.out.println("Name = " + name);
  System.out.println("Balance = "+balance);
}
```

Extending the Account class

```
class SAccount extends Account
{
   public SAccount(String accountID,String
      accountName,double amount,double minAmount){
      super(accountID, accountName, amount);
      this.minAmount = minAmount;
   }

   public SAccount(String accountID,
      String accountName, double amount)    {
      this(accountID,accountName,amount,0.0);
   }

   public double getMinAmount() {
         return minAmount;
   }
```

Two constructors

Accessor for subclass instance variable

```
    public void addInterest(double rate)    {
        deposit(getBalance() * rate/100);
    }


    void print()    {
        super.print();
        System.out.println("Min. Amount = " + minAmount);
    }

    public boolean withdraw(double amount) {
        if (getBalance() >= amount + minAmount)    {
              super.withdraw(amount);
              return true;
        }
        else return false;
    }

    private double minAmount;
}
```

New method

Overridden print()

calling superclass print()

Overridden withdraw()

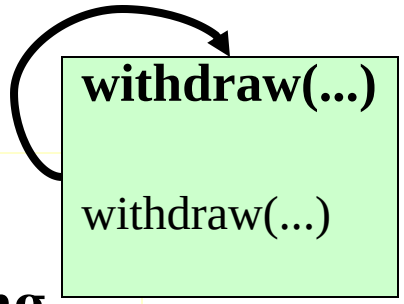
calling superclass withdraw()

subclass instance variable

What if …

What if I leave out super in super.withdraw(amount) ?

```
public boolean withdraw(double amount) {
      if (getBalance() >= amount + minAmount)    {
            super.withdraw(amount);
            return true;
      }
      else return false;
}
```



Result is a recursive call!

An infinite loop … cause program to hang

Which method is called ?

```
SAccount sAcc1 = new SAccount("s12345","Tim",1000,800);
```

```
sAcc1.deposit(100);
```
⟵ **Not overridden. Superclass method called**

```
sAcc1.withdraw(500);
```
⟵ **Overridden. Subclass method called**

```
sAcc1.addInterest(0.5);
```
⟵ **New method. Subclass method called**

Using the this reference

this refers to methods and instance variables of current object.

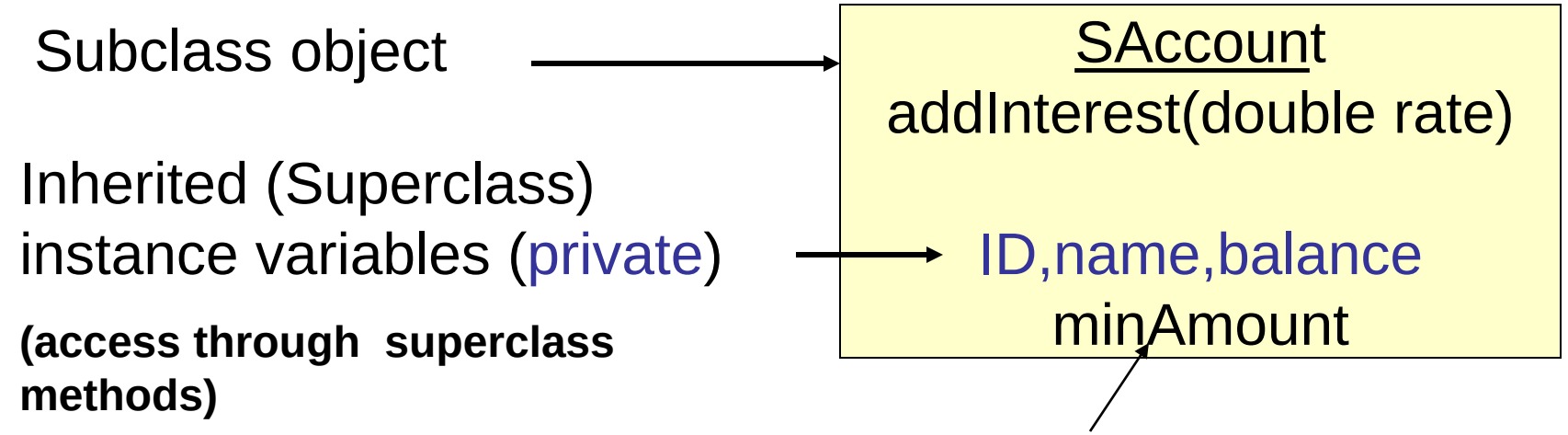
```
class SAccount extends Account
{

   public SAccount(String accountID, String
      accountName, double amount, double minAmount){
      super(accountID,accountName,amount);
      this.minAmount = minAmount;
   }

   . . .
   private double minAmount;
}
```

minAmount of current object (explicit)

Same name ?

Instance variable of superclasses

- Instance variables are automatically inherited by subclasses.
- But if they are private they cannot be accessed directly.
- Hence only way to change them is through superclass mutators.



What's wrong with SAccount withdraw() below ?

```
class SAccount extends Account
{
   public boolean withdraw(double amount) {
      if (balance >= amount + minAmount)
      {
            balance -= amount;
            return true;
      }
      else return false;
  }

    private double minAmount;
}
```

balance is a private instance variable of superclass Account !!!

Protected Access

- protected alternative to private and public
- If an instance variable is declared protected it can be accessed by methods of that *class,* its *subclasses* and all *other classes* within the same *package* (or *directory*).
- However, it cannot be accessed by other class users.
- Next Sample code illustrates the difference between the different *access specifiers*.

```
class A {
   …
   private int x;
   protected int y;
   public int z;
   void increment1() {
      x++;          // valid
      y++;          // valid
      z++;          // valid
   }
}
class B extends A {
   void increment2() {
      x++;          // invalid
      y++;          // valid
      z++;          // valid
   }
}
class SomeOtherClass {  // not in same package
   void increment3() {
      …
      A a = new A(…);
      a.x++;        // invalid
      a.y++;        // invalid
      a.z++;        // valid
   }
}
```

To use protected or not ?

Many programmers use the protected feature as it strikes a balance between *absolute protection* and *no protection* at all.

However, they break the *encapsulation rule* as the designer of the *superclass* has no control over the authors of the *subclass*.

Furthermore, classes with protected data are hard to modify as someone may have written a subclass based on it and may have accessed the protected data directly.

Quiz The Date class designed by *programmer A* was used by *programmer B* in class Meeting. Now B claims A's Date class has bugs. Comment.

```
class Date {   // Designed by programmer A
  Date(int d, int m, int y) {
    day = d;  month = m; year = y;
  }
  // advances d days taking care of month and year
  void advance(int d) {
    // ...
  }
  void print() {
          System.out.println(" "+day +"/"+month+"/"+year);
  }
  protected int day;
  protected int month;
  protected int year;
}
```

```
public class Meeting  { // designed by programmer B
    Meeting(String c, String t, Date d, String v)      {
                chairman = c; title = t;  date = d; venue = v;
    }
   void postpone(int days) {
                        date.day += days;
   }
   void print() {
                System.out.print(title+" "+chairman+ " "+venue);
                date.print();
   }
   public static void main(String arg[]) {
                        Date d = new Date(28,8,2001);
                        Meeting meet=new Meeting("Tom","AGM",d,"9.8");
                        meet.postpone(7);
                        meet.print();
   }
   private String title;
   private String chairman;
   private String venue;
   private Date date;
}
```

Composition Relationship

Subclass Construction

whenever a subclass object is constructed, the *superclass constructor* must be called.

syntax used: keyword super followed by construction parameters if any

must be the first statement in the method

If we omit this statement compiler looks for a *subclass* constructor with no arguments – *default constructor*.

```
public SAccount(String accountID, String
 accountName,double amount,double minAmount){
      super(accountID,accountName,amount);
         this.minAmount = minAmount;
}
```

Multiple Constructors

Suppose we have a Fraction class with two instance variables numerator and denominator. Naturally the constructor will take two arguments to set these instance variables.

To create a Fraction 3/4 we can call new Fraction(3,4)

How about creating 3 (same as 3/1) ? New Fraction(3,1)

But the designers of Fraction class may provide another constructor which takes only one argument the value for numerator making it convenient for users - new Fraction(3)

```
public Fraction (int num, int den)
{
    numerator = num;
    denominator = den
```

Standard constructor for Fraction class

```
public Fraction (int num) {
    numerator = num;
    denominator = 1
```

2nd constructor for Fraction class with only 1 argument, where denominator defaults to 1.

SAccount Multiple Constructors

SAccount = new SAccount(“s1234”,”Sam”, 120, 100);

SAccount = new SAccount(“s1235”,”Tom”, 3500, 0);

SAccount = new SAccount(“s1236”,”Tom”, 4200, 0);

SAccount = new SAccount(“s1237”,”Tom”, 3800, 0);

Most accounts have minimum amount set to 0.

Why not provide a constructor which will set it to 0 if no value is passed for min Amount ?

We can provide an additional constructor that will use the services of the first one with value for minAmount set to 0.

```
// note only 3 arguments are passed
public SAccount(String accountID,
      String accountName, double amount){
   this(accountID,accountName,amount,0.0);
}
```

Calls the existing SAccount constructor with 4 arguments.

Method Overloading

- Java allows two or more methods to have the same name provided the methods vary in the type of arguments or number of arguments is different.
- The actual method invoked is determined based on argument passed.
- This is known as method overloading.
- It is useful when we do similar operations on different type of attributes
- Consider a Member (library) class. We may want to update phone, fine or both address and phone using a method with the same name.

Method Overloading - Example

```
class Member
{ private String phone;
  private String address;
  private double fine;
  private Date expiry; // User written class
  // Overloaded update methods
  public update(String newFine){ … }  // 1st
  public update(Date newExpiry){ … }  // 2nd
  public update(String newPhone ){ … } // 3rd
  public update(String newAddress,String newAddress){ … }// 4th
  …
}
```

```
   // class user
   …
   Member m = new Member(…);
   m.update(new Date(…)); // invokes 2nd update
   m.update("98701235");  // invokes 3rd update
   m.update(12.50);       // invokes 1st update
   m.update("12 Kiwi rd, Kew" ,"96756734"); //invokes 4th update
   …
```

Quiz

Which of the statement in main() will result in compilation error?

Why?

What will be the output when that statement is removed?

```
class A
{  public void method1()
   { System.out.println("AAA");   }
}
class B extends A
{  public void method1()
   {   super.method1();
       System.out.println("BBB");
   }
}
class C extends B
{  public void method2()
   {  System.out.println("CCC");    }
}
public class Inherit
{  public static void main(String args[])
   {
      C c = new C();
      c.method2();
      c.method1();
      B b = new B();
      b.method2();
      b.method1();
   }
}
```

Quiz

What will be the output of the program below?

Why?

```
class A
{  public A()
   { System.out.println("AAA");
   }
}
class B extends A
{  public B()
   {  System.out.println("BBB");
   }
}
class C extends B
{  public C()
   {  System.out.println("CCC");
   }
}
public class Inherit2
{  public static void main(String args[])
   {  C c = new C();
   }
}
```

Quiz

Which one of the following methods cannot exist together within the same class ?

I public Account doSomething(Part p){..

II public Account doSomething(Account a){..

III public Part get(String ID){..

IV public Account get(String ID){..

(a) I and II

(b) I and III

(c) III and IV

(d) II and IV

Exercise: Creating a subclass of Part

- The part suppliers are beginning to face much competition and have decided to create a special class of parts (for slow moving parts) that can be given varying percentage of discounts, provided the total purchase cost exceed $1000.
- To cater for this requirement derive a subclass of Part named DiscountPart.
- Add an additional instance variable rate, for storing rate of discount for that specific DiscountPart.
- Provide a mutator (to change) and an accessor for rate.
- The constructor for this class should take an additional argument – rate of discount (applicable when sales >$1000)
- Override supply() method to return the discounted price. This method should first call the superclass supply method to compute the original price. It should also override the print() method to print the rate of discount.
- Write a driver class to test the class.

Exercise: Creating a subclass of Part

Exercise: Creating a subclass of Part